Patrón Estructural Decorator

Universidad del Cauca

Vigilada Mineducación

Laboratorio de Ingeniería de Software II

Practica de Laboratorio No. 6

Presentado por:

Jeferson Castaño Ossa

David Santiago Giron Muñoz

Profesor:

Ricardo Antonio Zambrano Segura

Universidad del Cauca

Facultad de Ingeniería Electrónica y Telecomunicaciones

Ingeniería de Sistemas

Popayán, Octubre de 2023

Patrón Estructural Decorator

Intención:

La intención del patrón decorador es brindar una mejor y más flexible alternativa para la extensión de funcionalidades de un sistema de manera dinámica sin necesidad de modificar su código estructural principal. De esta manera, juega el papel de un decorador, al ir adicionando y envolviendo el “Funcionamiento/objeto base” en otras responsabilidades o especificaciones alrededor de él recursivamente, es decir en cuantas “capas de envoltura” se quiera.

Problema que soluciona:

En algunas ocasiones se desea expandir las funcionalidades de ciertos objetos, de tal manera que no solo aplique su comportamiento ya existente, sino que incorpore uno adicional. El modificar la clase principal no es una opción, ya que no se desea añadir esta responsabilidad a todos los objetos de la clase, sino sólo a unos individuales y crear varias subclases que implementan estas funcionalidades de manera combinada no es viable ya que incrementaría enormemente el número de subclases y el tamaño del código de estas.

De esta manera, el patrón decorador busca resolver el problema estructural de añadir comportamientos o estados a objetos de manera dinámica sin aplicar la extensibilidad por herencia ya que modificaría su estructura original.

Solución propuesta:

El patrón Decorator da solución al problema de agregar responsabilidades adicionales a un objeto de forma dinámica sin alterar su estructura, gracias a uno de los principios básicos de la programación orientada a objetos: favorecer la composición por sobre la herencia. En lugar de crear una jerarquía de subclases para heredar comportamientos de superclases y crear herencia múltiple para conseguir cada una de las combinaciones de las características deseadas, el patrón Decorator utiliza una estructura de clases que permite que los objetos se "envuelvan" con **decoradores,** los cuales agregan responsabilidades adicionales al objeto envuelto. Cada decorador es responsable de una nueva característica específica.

Si se desea agregar alguna funcionalidad adicional a algún objeto, ya no se modifica directamente la clase original, ahora solo se debe crear un decorador concreto que hereda del **Decorador** y este es el que envuelve al objeto original. Como los decoradores y los objetos implementan una abstracción en común, el patrón permite

que se apilen decoraciones en cadena, brindando la posibilidad de agregar un sinfín de características especiales al objeto núcleo a decorar en tiempo de ejecución.

En resumen, la solución que propone el patrón Decorator se basa en la composición de objetos y la encapsulación de responsabilidades adicionales en decoradores que envuelven el objeto original. La estructura del patrón y sus respectivos participantes se verán más a fondo a continuación.

Diagrama de clases:

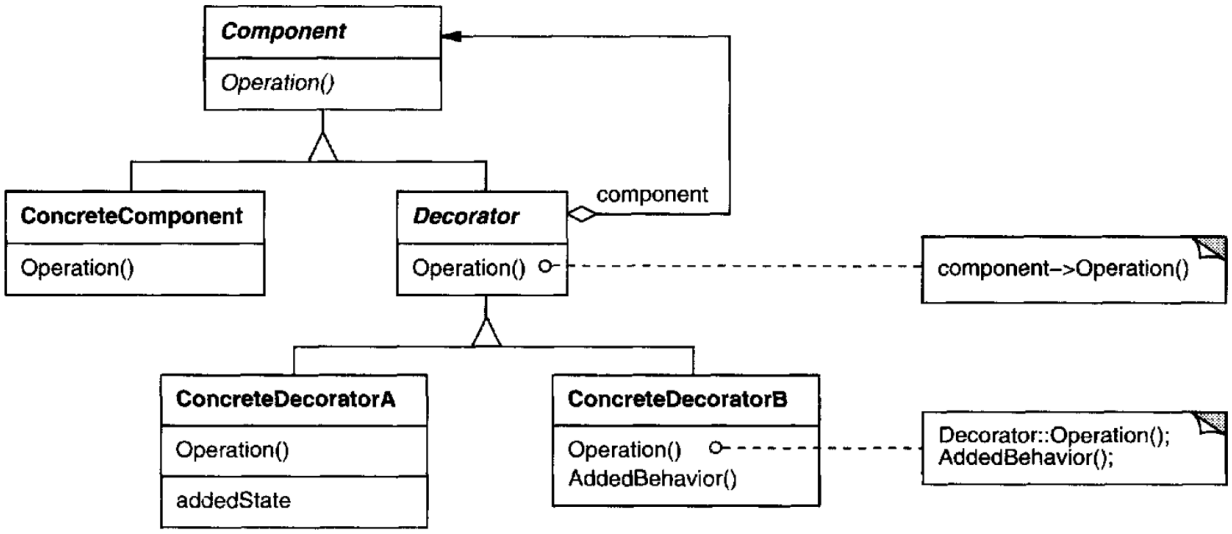
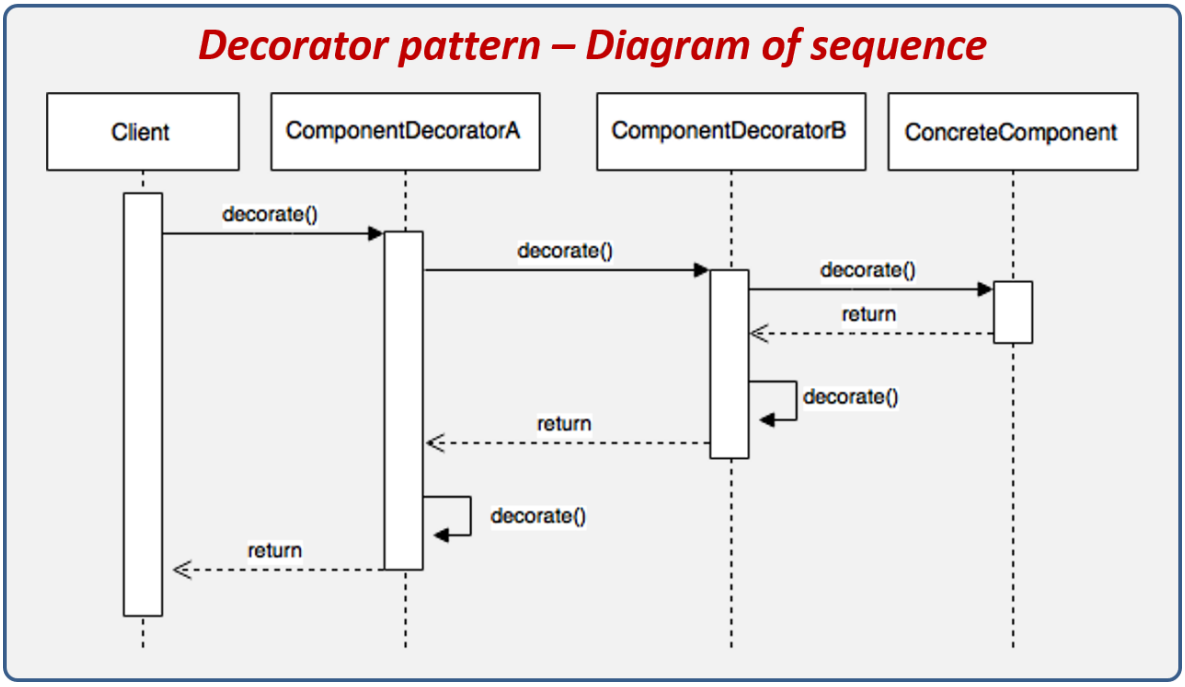


Diagrama de secuencia:



Participantes:

- Componente:

En esta interfaz se definen tanto los comportamientos que son implementados por los componentes concretos (objetos envueltos), como por los decoradores (envoltorios). De esta manera, define la interfaz para los objetos que pueden tener responsabilidades añadidas a ellos.

- Componente concreto:

Implementa la clase componente y define la clase de objetos núcleos (envueltos), los cuales pueden ser decorados. Es decir, son objetos los cuales brindan un comportamiento básico que puede ser complementado por las responsabilidades adicionales adjuntas de los decoradores.

- Decorador:

Es la clase abstracta para todos los decoradores. En esta se define una referencia al componente original, ya que todas las funcionalidades las delega al objeto envuelto. También implementa la interfaz componente, ya que la interfaz para los decoradores concretos que van a actuar como una envoltura, debe coincidir con la interfaz del componente.

- Decorador concreto:

Extiende de la clase decorador para agregar las responsabilidades adicionales al componente concreto. Es decir, definen los objetos en los cuales los componentes pueden estar envueltos, combinando varias características. Pueden llamar a las operaciones del componente base antes o después de su comportamiento específico.

Aplicabilidad:

El patrón decorator se puede aplicar cuando:

- Se desee añadir responsabilidades a objetos individuales, de manera dinámica y transparente sin llegar a afectar otros objetos por tener que modificar el código.
- Responsabilidades de un objeto que pueden ser retiradas, cuando todos los objetos que implementan un supertipo no deben tener una responsabilidad.
- Se quiere evitar una jerarquía de clases llena de subclases para cada combinación de responsabilidades.

- Cuando una herencia es impráctica. Esto es cuando un gran número de extensiones son posibles, pero éstas producen una infinidad de subclases para poder soportar cada una de las combinaciones. O sencillamente cuando un objeto no puede heredar de una clase en concreto para aumentar su responsabilidad.

Consecuencias:

Las consecuencias del patrón Decorator son:

- Flexibilidad y extensibilidad: Puedes agregar o quitar responsabilidades dinámicamente mediante la composición de decoradores. Esto facilita la extensión de funcionalidades sin modificar clases existentes. Del mismo modo, permite la mezcla de combinaciones de las responsabilidades en tiempo de ejecución.

- Evita la sobrecarga de subclases: En lugar de crear muchas subclases para cada combinación de características, puedes usar decoradores para construir combinaciones personalizadas.

- Alto nivel de complejidad: El uso excesivo de decoradores puede llevar a una alta complejidad en la configuración de objetos y dificultar la comprensión del sistema.

- Puede haber un impacto en el rendimiento: La anidación de decoradores puede afectar el rendimiento. Esto a raíz de que se generan un gran número de pequeños objetos que son similares entre sí, cambiando solo la manera en que son interconectados. Esto último hace que sea difícil entenderlos y el debug del código.